On the Description of the Communication Protocol HDLC in cHFP

Miyachi, Toshio; Katayama, Takuya

数理解析研究所講究録
586巻 1986年 182-195

On the Description of the Communication Protocol HDLC in cHFP

東京工大 情報工学科　宮地 利雄 (Toshio Miyachi)
片山 卓也 (Takuya Katayama)

Department of Computer Science,
Tokyo Institute of Technology,
2-12-1 O-okayama, Meguro, Tokyo, Japan 152

1. Introduction

This paper presents a description of communication protocol HDLC in cHFP as an example of cHFP application to an complex concurrent processing system like communication programs. The functional concurrent computation model cHFP is an extended descendant of HFP which was based on the attribute grammar and module hierarchy. It enables us to describe concurrent processes performing nondeterministic actions and synchronizations in the functional language framework, we can enjoy description on harmonious combination between data-flow principle and sequential flow of control.

A number of methods for formal specification and verification of communication protocols have been proposed[1,2]. It is a hard problem to give formal specification of communication systems, since they includes concurrency and nondeterminism. Of course the specification method must be easy to analyze as well as to implement.

This paper is organized as follows : in section 2, we outline cHFP; in section 3, the description of HDLC in cHFP is given after brief introduction of HDLC; in section 4, we make reference to relating works and draw future plans of further work as concluding remarks.

2. Functional concurrent computation model : cHFP

The functional concurrent computation model **cHFP**[3,4] is an extended descendant of **HFP** which was proposed by T. Katayama[5] as a hierarchical functional language based on the attribute grammar and the module hierarchy. It enables us to describe concurrent processes performing nondeterministic actions and synchronizations in the functional language framework by introducing rendezvous together with synchronizing and sequential condition. We can enjoy on harmonious combination between the data-flow principle and the sequential flow of control.

A **cHFP** program is composed of a number of cuncurrently executed processes. A process of **cHFP** is an actor which extend a computation tree and evaluate attribute values on its nodes. Processes can communicate and synchronize each other only by binding leaf nodes of their computation tree. The binded special type nodes are called communication ports, and the others are called modules.

In the same way as a set of derivation trees is defined in formal grammar, formally **cHFP** program is described as a 4-tuple :

(**M**, **C**, **P**, **Er**)

where **M** is a set of modules, **C** is a set of communication ports. The nodes of computation trees are members of $M \cap C$. For any element c of **C** there exists an unique element $\bar{c}$ in **C** which is called a <u>complementary communication port</u> to c, and $\bar{\bar{c}} = c$. For every element n of **M** and **C** two mutually disjoint sets I(n) and O(n) are associated. The elements of I(n) and O(n) are called input attributes and output attributes of n, respectively. Input and output attributes are called simply 'attributes'. They work like variables in conventional programming lan-

guages. We assume that $I(c)=O(\bar{c})$ and $O(c)=I(\bar{c})$ for every communication port $c \in C$. P is a set of processes. Each process $p \in P$ has a unique initial module i_p such that $i_p \in M$ and $I(i_p)=\phi$.

Er is a set of extension rules. According to Er, every process extends its computation tree, which is initially a trivial tree with the initial module i_p as its root. The general form of extension rules is as follows :

$X_0 \rightarrow X_1 , \cdots , X_t;$

<attribute-value-definition>

when <extension-condition>

synch <synchronization-condition>

order <sequence-condition>

where $X_0 \in M$ is called left-side of the extension rule, and a list X_1, ..., X_t of elements of $M \cup C$ is called right-side of the extension rule. Right-side of an extension rule may be empty list. When the same element of $M \cup C$ appears twice or more in an extension rule, we annex an appropriate suffix to them in order to identify each module or communication port which is refered an occurence.

The attribute-value-definition is a set of equations, which determine the value of each attribute that belongs to $O(X_0)$ or $I(X_i)$ from the attribute values that belongs to $I(X_0)$ or $O(X_i)$, where $1 \leq i \leq t$. In contrast to assignment to variables of conventional languages, cHFP evaluates each attribute value only once by pure functions with no side effect. So the result of cHFP program does not depend on the order of evaluation of attribute values.

The three conditions in extension rules are optional. The extension condition is a predicate over attributes belonging to $I(X_0)$. Whenever the extension rule is applied, its extension condition must be

true. The synchronizing condition is a subset of set { X_1, ... ,X_t } ∩ C, that is, instances of communication ports which are included in right-side of the extension rule. We can apply the extension rule, only if immediately after applying the rule, every instance of communication ports specified in the synchronizing condition can be connected to its complementary instance in other process (This connection between complementary communication ports are called 'rendezvous') as soon as they are created. The sequencing condition is a partial ordered relation over the set {X_1, ... ,X_t} of occurences in right-side, and the order of rendezvous of communication port instances mustn't contradict the partial ordered relation induced by the sequencing conditions.

The computation process of **cHFP** goes on along three axes, i.e. extension of computaion trees, evaluation of attribute value on nodes of the computation trees, and rendezvous between a pair of communication ports. Computaion trees are formed in the way similar to the derivation trees in formal language theory, in which extension rules, modules, communication ports, and an initial module work like rules of grammar, nonterminal symbols, terminal symbols, a start symbol, respectively. That is, if a module exists at leaf of a computation tree and also is left-side of an applicable extension rule, the new instances of right-side of the extension rule are created, and added to the computation tree as sons of them. The construction of computation trees begins at trivial ones consisting of an initial module of processes, and is repeated while there are any applicable extension rules.

The values of attributes on instances of modules and communication ports is undefined when they are newly created. The attribute-value-definition part of extension rule specifies how the attribute values are defined. This evaluation process is completely data-driven with no

side effects. Although Id[6] and Concurrent Prolog[7] essentially need variables of stream type since communication between processes are expressed with streams, whose partial value is determined step by step during computation, in our model employs no stream-type value.

Rendezvous is an action that a pair of communication port instances are connected and exchange their attribute values. It is only means for inter-process communications in our model. It is at most one rendezvous the each communication port instance is engaded to.

From the viewpoint of object oriented languages, processes of **cHFP** are objects, and communication ports can be regarded as methods of objects. In **cHFP**, the state of objects is defined by computation trees and values of attributes on nodes of computaion trees, and the method bodies of objects are defined by the extension rules. This distinguishes the object oriented programming in **cHFP** from one in Smalltalk-80, whose method bodies are described as a sequentially executed command sequence whose components are called message-expressions. Rendwzvous of **cHFP** as inter-process communication is noteworthy on the following two viewpoints : One is that the evaluation of attribute values engaged in rendezvous is not necessary to have been completed at rendezvous time because **cHFP** rendezvous is only binding between attribute instances, while values are transmitted as messages in almost all message passing style inter-process communication scheme. Another point is that there is no syntactic distinction between sending and receiving operation of rendezvous, that is rendezvous between a pair of processes is carried out symetrically.

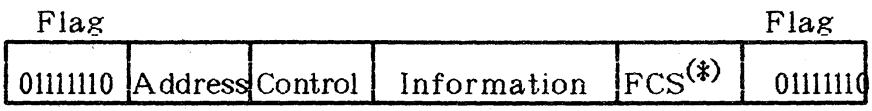
3. Description of the communication protocol HDLC in cHFP

3.1 Communication protocol HDLC

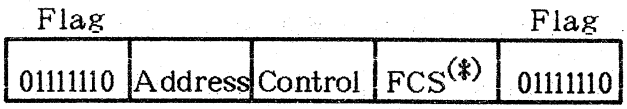
HDLC[8,9] is a communication protocol for data link control, and has been developed by ISO in response to IBM's SDLC protocol. HDLC permits us to transmit arbitrary bit sequences with variable length in point-to-point link or multi-point link network with high reliability by handling errors of physical layer. Also CCITT defines a subset of HDLC in its recommendation X.25[10] as the second layer (data link) protocol. The actions of HDLC communication system are separated into three stages : (1) initialization, (2) data exchange, and (3) closing session and releasing link. As for data exchange, three kinds of communication mode is supported, which are independent to each other. In this paper we show a part of the description of data exchange in Normal Response Mode (NRM) which is the most fundamental.

In NRM there are a priori specified one primary station and one or more secondary stations. Although data transmision can be initiated by any station, a primary station has a special role to watch the timer in order to detect and handle time-out errors.

Data and link control inforamtion are exchanged in the specific form called a frame. The frame consists of several fields. The format of frames has one of information transfer format (I), supervisory format (S), and unnumbered format (U). Fig.1 shows each of them. The command field in a frame specify the frame format and the function of the frame.



(a) Information transfer frame format.



(b) Supervisory and unnumbered frame format.

(*) FCS is a frame checking sequence.

Fig.1 the HDLC frame format

The control field in a frame carries inforamtion expressed in the following Pascal record data:

```
type control = record                    pf : boolean;
               case  flag : ISU  of
               I :   ( record  s,r : 0..7;  end );
               S :   ( record  cmd: Scommands;  r : 0..7;  end );
               U :   ( record  cmd: Ucommands  end )
               end
```

where the data s (r) is called a send (receive) sequence number, indicating its frame sequence number (the expected value of s field of a next I command), and inspected to check out frame sequence error (respectively).

In the range of this paper it is sufficient to indentify only three commands : informatioan command (I), receive ready command (RR), and receive not ready command (RNR). I command has I format, and its function is to transfer across a data link sequentially numbered frames containing an information field. RR and RNR commands have S format. RR command is used to (1) indicate it is ready to receive an I command; (2) acknowledge previously received I commands numbered up. RNR command is used to indicate a busy condition; i.e., temporary inability to accept additional incoming I commands. RR may be used to clear a busy condition that was initiated by the transmission of RNR.

3.2 Representation of system status

A HDLC point-to-point communication system is considered, which includes a pair of stations (a primary and secondary station), users ($user_p$ and $user_s$) for each station, up and down communication lines ($line_{up}$ and $line_{down}$), and a timer which alarms timeout errors (timer). It is modeled as a set of processes. To specify communication procedures is to define how a pair of stations interact with other processes, which constitute an environment of communication procedures by specifying how the stations change its state and which frame it sends and receives.

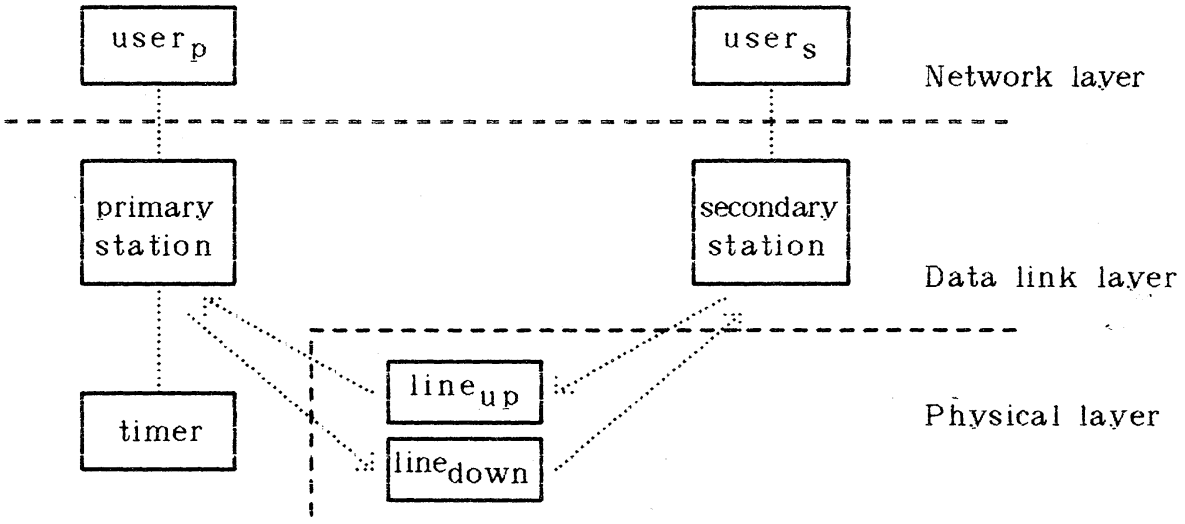


Fig.2 The communication system.

Since the computation of **cHFP** is defined as a sequence of extensions of computation trees, roughly speaking, the state of each station in represented by the attribute values on a certain module, say S, and a new instance of the module is created when the state is changed, since the attributes of **cHFP** can't be substituted more than twice. Subsidiary state such as error handling can be represented by introducing other modules.

A set of atributes of the module S is { rdata, sdata, seq, nextsnd,

nextrcv, ack, checkpoint, t-error, ready, busy }; where rdata and sdata are the buffers whose entries are accessed with sequencial number and which contains data to be sent and received respectively; seq is the send sequence number of data to be sent next; nextsnd is the send sequence number of data to be sent next for the first time (not one sent again for error recovery); nextrcv is the send sequence number of frame to be received next; ack is the receive sequence number of another station which has been already reported to this station; checkpoint is the send sequence number of the frame sent most recently with PF-bit on; t-error is true when timeout error is detected; ready is true when a frame with PF-bit on is received, i.e. when the PF-bit of a next issued frame may be on; busy is true while no response to RNR command has been received.

3.3 Representation of data exchange

Data exchange in a communication system, i.e. input and output from/to communication line, in other word, interaction with downward layer in layer protocol model, is repredented as rendezvous with the processes representing physical layer. In the **cHFP** description, six communication ports, which are refered by a pair of command name and action, e.g. I_s, RR_r and so, are introduced in order to model sending and receiving three kinds of commands, Fields in communication frames are modeled as attributes on communication ports.

The communication ports and their attributes which are intorduced for the HDLC description are shown in the following :

I_s : $I(I_s)=\{ s, r, d, pf \}$ $O(I_s)=\phi$

RR_s,RNR_s : $I(RR_s)=I(RNR_s)=\{ r, pf \}$ $O(RR_s)=O(RNR_s)=\phi$

I_r : $I(I_r)=\phi$ $O(I_r)=\{ s, r, d, pf \}$

RR_r,RNR_r : $I(RR_r)=O(RNR_r)=\phi$ $O(RR_r)=I(RNR_r)=\{$ r, pf $\}$

The values of data pf, s, r in the field control of frames are represented by the attributes which have a corresponding name. The attribute d contains the data in the information field of I commands.

Though attribute value may not be evaluated at rendezvous time in cHFP, they must be evaluable at rendezvous time if the implementation is executable.

3.4 Description of procedures

All we have to do in order to describe the communicaton procedures is to write down what frames are sent and received, and how stations change their states when sending or receiveing frames. In cHFP we can do this by giving the extension rules and specifying the creation of communication ports and modules and the definition of attribute values.

In the following as a fragment of the cHFP description of communication procedures we show the procedure executed when receiving and sending I command.

```
[[ receiving I command at the primary station ]]
  S0 → Ir, S1, if S0.ack ≠ S1.ack then TIMER:reset fi
        some-frame-rejected := Ir.pf and S0.checkpoint ≠ Ir.r and
                               order(S0.checkpoint, Ir.r, S0.ack);
        validIcommand := ( Ir.s = S0.nextrcv );
        S1.seq := if some-frame-rejected  then Ir.r  else S0.seq;
        S1.nextsnd := S0.nextsnd;
        S1.ack := Ir.r;
        S1.nextrcv := if validIcommand  then S0.nextrcv + 1
                                        else S0.nextrcv;
        S1.checkpoint := if order(Ir.r, S0.checkpoint, S0.ack)
                         then  Ir.r      else S0.checkpoint;
        S1.ready := S0.ready or Ir.pf;
```

S_1.rdata := **if** validIcommand **then** append(S_0.rdata,I_r.d)
else S_0.rdata;
S_1.t-error := S_0.t-error; S_1.busy := S_0.busy;
synch I_r
order **if** S_0.ack ≠ S_1.ack **then** I_r << TIMER:reset << S_1
else I_r << S_1 **fi**

In the above description, order(s,t,u) is a predicate to become true when t is greater or equal to s, t is less or equal to u, and s is not equal to u in modulo 8 numbering system.

The extension rule given above represents the behaviour of the primary station receiving a I command ; it changes its state from S_0 to S_1, and then reset a timer. Two local variables some-frame-rejected and validIcommand are used in order to simplify the description, and expressing that some frames are discarded in another station because of detected error, and that another staion is judged to be able to receive commands, respectively. The detail of the state transition is specified by defining the attribute values of S_1 in terms of those of S_0. That is, this station gets data of the information field of the received frame, if its s field has same value as nextrcv. And it also know that the other station had rejected or not received some frames which this station sent before, and that they should be sent again, if some-frame-rejected is evaluated true.

In the above rule the description **if** S_0.ack≠S_1.ack **then** TIMER:reset **fi** is a syntax sugar meaning that an instance of the communication port TIMER:reset is also created if and only if the specified condition holds.

[[sending I command from the primary station]]
S_0 → I_s, **if** timer is not running **then** TIMER:start **fi**, S_1

I_S.s := S_0.seq; I_S.r := S_0.nextrcv;
I_S.sdata := S_0.sdata[S_0.seq];
if S_0.ready **then** I_S.pf := **true**;
S_1.seq := S_0.seq + 1;
S_1.nextsnd := **if** S_0.seq = S_0.nextsnd **then** S_1.seq
else S_0.nextsnd;
S_1.nextrcv := S_1.nextrcv; S_1.ack := S_0.ack;
S_1.checkpoint := **if** I_S.pf **then** S_0.seq **else** S_0.checkpoint;
S_1.ready := S_0.ready **and not** I_S.pf;
S_1.t-error := S_0.t-error; S_1.busy := S_0.busy;
when S_0.sdata[S_0.seq] is available **and** S_0.seq + 1 ≠ S_0.ack
order I_S << S_0

Above rule represents the behaviour of the primary station sending a I command ; if there is data to be sent and the number of data which this station has already sent and which haven't been received by the other station doesn't exceed 8, it sends a I command, changes its state from S_0 to S_1, and then start a timer if the timer has not been started.

4. Discussion and conclusion

Methods of protocol specification may be classified into three main categories : state transition models, programming languages, and combinations of the first two. Generally speaking, specification in a state transition model is rather easy to analyze, while it is hard to implement, and one in the other model is hard to analyze, while it is easy to implement.

Though our method described in this paper falls into the programming language approach, it makes the situation better since it has featurers of functional language. T. Katayama et. al.[11] showed a verification method of attribute grammar on which our computation model **cHFP** is based. T. Kasami et. al. showed an algebraic specification of HDLC

and verified some of its properties in [2]. Our description resembles to theirs very well, so it is expected that a corresponding verification technique can be developed.

L. H. Landweber et. al.[12,13] works about protocol specification by attribute grammar. They regards communication events such as sending and receiving frames as terminal symbols of the attribute grammar, and interpret statements generated by the attribute grammar as histories of communication. We are working on bridging between our approach and theirs.

[References]

[1] S. S. Lam and A. U. Shankar : "Protocol Verification via Projections", IEEE Transactions on Software Engineering, Vol.SE-10, No.4 (1984)

[2] T. Higashino, M. Mori, Y. Sugiyama, K. Taniguchi and T. Kasami : "An Algebraic Specification of HDLC Procedures and Its Verification", IEEE Transactions on Software Engineering, Vol.SE-10, No.6 (1984)

[3] T. Miyachi, T. Katayama : "On the capability of hierarchical functional computation model", Japan Society for Software Science and Technology, 1st Conference. [in Japanese] (1984)

[4] T. Miyachi, T. Katayama : "**concurrent HFP** : a functional computational model for parallel processing", Tokyo Inst.of Tech. Dept.of Computer Science T.R. CS84-TM03. (Oct.1984) Its Japanese version is on Transacton of Information Processing Society of Japan Vol.27, No.1, (1986)

[5] T. Katayama : "A Hierarchical and Functional Programming Based on Attribute Grammar", 5th Int. Conf. on Software Engineering (1981)

[6] K. P. Arvind and W. P. Gostelow : The (prelimianry) Id Report : An Asynchronous Programming Language and Computing Machine, Tech.Rep. 114, Dept. of Comp. Sci., Univ. of California, Irvine, (1978)

[7] E. Y. Shapiro : A subset of Concurrent Prolog and Its Interpreter,

ICOT Tech. Rep. TR-003, (1982)

[8] ISO : "Data Communication -- High Level Data Link Control Procedures -- Frame Structure", International Standard ISO 3309, (1979)

[9] ISO : "Data Communication -- High Level Data Link Control Procedures -- Elements of Procedures", International Standard ISO 4335-1979(E), (1979)

[10] CCITT : CCITT Recommendation Interface between Data Terminal Equipment and Data Circuit Terminating Equipment for Terminals Operatinf in the Packet Mode on Public Data Networks, Fascicle VIII.2 -- Rec. X.25, (1980)

[11] T. Katayama and Y. Hoshino : "Verification of Attribute Grammars", Proc. 8th ACM Symp. of Principles of Programming Languages (1981)

[12] D. P. Anderson and L. H. Landweber : "A Grammar Based Methodology for Protocol Specification and Implementation", Univ. of Wisconsin, Computer Science Dept. Tech. Rep." 608 (1985)

[13] D. P. Anderson : "A Grammar Based Methodology for Protocol Specification and Implementation", Univ. of Wisconsin, Computer Science Dept. Tech. Rep." 612 (1985)